CS360 Midterm #2 – Spring, 2014 – James S. Plank – March 11

In all questions, assume that pointers are 4 bytes, and that the machine architecture is like **jassem**.

Question 1

```
typedef unsigned int UI;

void pm(double *p)
{
  char **s;
  char *x;
  int *ip;
  int i;

  s = (char **) p;
  p++;
  ip = (int *) p;

  printf("1. 0x%x\n", (UI) s);
  printf("2. %s\n", s[0]);
  printf("3. %s\n", s[4]);
  printf("4. %d\n", *ip);
  printf("5. %d\n", ip[1]);

  x = (char *) s;
  for (i = 0; i < 4; i++) {
    s[1][i] = *x;
    x += 4;
  }
  printf("6. %s\n", s[1]);
  printf("7. 0x%x\n", (0x7c54 >> 2));
  printf("8. 0x%x\n", (0xa45e << 2));
}
```

```
Address:         Decimal        Hex               Chars
0x572b20        5712724     0x00572b54     'T' |'+' |'W' |'\0'
0x572b24        5712688     0x00572b30     'O' |'+' |'W' |'\0'
0x572b28        5712716     0x00572b4c     'L' |'+' |'W' |'\0'
0x572b2c        5712696     0x00572b38     '8' |'+' |'W' |'\0'
0x572b30        5712708     0x00572b44     'D' |'+' |'W' |'\0'
0x572b34        5712700     0x00572b3c     '<' |'+' |'W' |'\0'
0x572b38        5712704     0x00572b40     '@' |'+' |'W' |'\0'
0x572b3c        5712676     0x00572b24     '$' |'+' |'W' |'\0'
0x572b40        5712680     0x00572b28     '(' |'+' |'W' |'\0'
0x572b44        5712712     0x00572b48     'H' |'+' |'W' |'\0'
0x572b48        5712692     0x00572b34     '4' |'+' |'W' |'\0'
0x572b4c        5712720     0x00572b50     'P' |'+' |'W' |'\0'
0x572b50        5712672     0x00572b20     ' ' |'+' |'W' |'\0'
0x572b54        5712728     0x00572b58     'X' |'+' |'W' |'\0'
0x572b58        5712684     0x00572b2c     ',' |'+' |'W' |'\0'

p is 0x572b20.
```

You know the drill. Tell me the output of the procedure given that state of memory.

Question 2

Suppose I have a C procedure called **a()**, whose first few lines are depicted to the right. Those are the only variable declarations in **a()**. When it is called, the **sp** and **fp** have values of `0xffff440`. Later during its execution (but still in **a()**), the 28 bytes of memory starting at address `0xffff428` are depicted on the right.

```
int a(int i, char *s)
{
   int b[2];
   int *x;
   int p;

   ...
```

0xffff428	0xffff43c
0xffff430	0xffff428
0xffff434	0x10420
0xffff438	0xffff444
0xffff43c	0xffff44c
0xffff440	0x10400
0xffff444	0xffff460
0xffff448	0x10448
0xffff44c	0xffff450
0xffff450	0xffff45c

Please answer the following questions – give all values in hex, except for part F, which asks for an instruction).

A. What is the address of **p**? (have your compiler work as in class, with **b**, **x** and **p** being stored in successively higher addresses).
B. What is the address of **x**?
C. What is the address of **b[0]**?
D. What is the address of **i**?
E. What is the value of **i**?
F. What is the first assembly code instruction of "a:"?
G. What was the frame pointer of the procedure that called **a()**?
H. What is the memory address of the **jsr** instruction that called **a()**?
I. What is the address of **s**?
J. What is the value of **p** in hex?
K. What is the address of **s[0]** (in other words, what is **&(s[1])**)?
L. What is the address of **s[1]** (in other words, what is **&(s[0])**)?
M. What is ***x**?
N. If we have to spill **r2**, what is the address of the memory where it will be spilled?
O. If I do `printf("0x%x\n", b[3])`, what will it print?

Question 3:

Don't bother with include statements in your code.

Write a program that prints the filenames of all files and directories in the current directory. If multiple file names are links to the same file, print only one of them. Print the filenames in any order that you want. Don't worry about symbolic links. You may not call **realpath()** either, because that is a revolting system call. If you ignore links, and just print out all of the files in the current directory, you will only receive half credit.

Question 4 (which will be worth more points than question 3):

Question 4 pertains to the files `r16.c` and `r16.h`, which are on the next page.

Part A: Write **`cat`** (a program that prints standard input on standard output) using only **printf()** and the procedures defined in `r16.c` and `r16.h`.

Part B: Suppose we changed `SIZE` to 64. Would you expect your `cat` to run faster or slower on a large file? Explain the reason why.

Part C: Suppose we wrote `cat` as follows (assume all of the includes are correct):

```
main()
{
  char buf[16];
  int i;

  while (1) {
    i = fread(buf, 1, 15, stdin);
    if (i == 0) exit(0);
    buf[i] = '\0';
    printf("%s", buf);
  }
}
```

Would you expect this `cat` to be faster or slower than the `cat` in Part A? By a lot or a little? Explain why.

Part D: Give me a specific example of an input file where the `cat` in Part A will not output the exact contents of standard input.

Part E: Suppose you pass a bad file descriptor to `R16_setup()`, and then you call `R16_read()` with a buffer that is larger than 17 chars. What is going to happen?

Helpful prototypes and typedefs

```
DIR *opendir(char *dir);
struct dirent readdir(DIR *d);
int stat(char *filename, struct stat *buf);
JRB make_jrb();
JRB jrb_insert_int(JRB t, int key, Jval val);
JRB jrb_insert_str(JRB t, char *key, Jval val);
JRB jrb_find_int(JRB t, int key);
JRB jrb_find_str(JRB t, char *key, Jval val);
Dllist new_dllist();
Dllist dll_append(Dllist d, Jval val);
```

```
struct stat {
    dev_t    st_dev;
    ino_t    st_ino;
    mode_t   st_mode;
    nlink_t  st_nlink;
    uid_t    st_uid;
    gid_t    st_gid;
    dev_t    st_rdev;
    struct timespec st_atimespec;
    struct timespec st_mtimespec;
    struct timespec st_ctimespec;
    off_t    st_size;
};
```

```
/* r16.c */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "r16.h"

#define SIZE 4096

typedef struct {
  char b[SIZE];
  int eof;
  int p;
  int fd;
} R16_t;

void *R16_setup(int fd)
{
  R16_t *r;

  r = (R16_t *) malloc(sizeof(R16_t));
  r->eof = -1;
  r->p = SIZE;
  r->fd = fd;
  return (void *) r;
}

void R16_read(void *r16, char *buf)
{
  R16_t *r;
  int bytes, i;

  r = (R16_t *) r16;

  if (r->p == SIZE) {
    r->eof = read(r->fd, r->b, SIZE);
    if (r->eof == SIZE) r->eof = -1;
    r->p = 0;
  }

  bytes = 16;
  if (r->eof != -1 && r->p+16 > r->eof) {
    bytes = r->eof - r->p;
  }
  for (i = 0; i < bytes; i++) {
    buf[i] = r->b[r->p+i];
  }
  buf[bytes] = '\0';
  r->p += bytes;
}

void R16_jettison(void *r16)
{
  R16_t *r;

  r = (R16_t *) r16;
  free(r);
}
```

```
/* r16.h */
extern void *R16_setup(int fd);
extern void  R16_read(void *r16, char *buf);
extern void  R16_jettison(void *r16);
```